



Pontte Coding Challenge

Documentação

Histórico de Revisões

Versão	Data	Responsável	Comentário
1.0	08/01/2020	Danilo Paggi	Criação do documento.

Sumário

1. Tecnologias utilizadas.....	5
2. Pré-requisitos	5
2.1. PostgreSQL	5
2.2. Node	5
2.3. Gerenciador de pacote	5
2.4. API Client	5
3. Configurações.....	5
3.1. Configurando a aplicação.....	6
3.2. Configurando o banco de dados	6
3.3. Configurando o Insomnia (API Client)	6
4. Scripts.....	7
4.1. Carregando as bibliotecas.....	7
4.2. Preparando a base de dados.....	8
4.2.1. Criando as tabelas.....	8
4.2.2. Criando os registros iniciais.....	8
4.3. Rodando a aplicação.....	8
4.4. Rodando os testes	9
4.4.1. Linux	9
4.4.2. Windows.....	9
5. Organização das pastas do projeto	9
5.1. <Pasta raiz>.....	10
5.2. __tests__	10
5.3. integration.....	10
5.4. unity	10
5.5. docs	10
5.6. node_modules.....	10
5.7. src.....	10
5.8. app.....	10
5.9. controllers	10
5.10. exceptions.....	10
5.11. functions	10
5.12. models	10
5.13. validators.....	10

5.14.	config	11
5.15.	database	11
5.16.	migrations	11
5.17.	seeders	11
5.18.	uploads	11
6.	Banco de dados da aplicação	11
6.1.	SequelizeMeta	11
6.2.	contracts	11
6.3.	documents	12
6.1.	document_types	12
6.2.	contract_steps	12
6.3.	marital_statuses	12
7.	Endpoints	13
7.1.	Retorna a lista com os tipos de documento	13
7.2.	Retorna a lista de estados civis	13
7.3.	Retorna a lista de etapas de um contrato.	13
7.4.	Retorna o contrato correspondente a seu código de identificação.	14
7.5.	Lista os contratos vinculados a um cliente pelo seu CPF	14
7.6.	Cria um novo contrato de empréstimo	15
7.7.	Altera um contrato de empréstimo já existente	16
7.8.	Envia documentos referentes a um contrato de empréstimo	17
7.9.	Aprova ou desaprova o contrato de empréstimo	18
8.	Melhorias	18

1. Tecnologias utilizadas

A stack utilizada no projeto seguiu conforme abaixo:

- NodeJs (v.10.16.3) nos módulos referentes a API.
- PostgreSQL (v. 12.1) no banco de dados.

2. Pré-requisitos

É necessário que o ambiente aonde será rodado o projeto tenha instalado os seguintes componentes: PostgreSQL, Node e um gerenciador de pacotes (NPM ou Yarn). Abaixo seguem os sites oficiais e tutoriais de instalação de cada componente em cada sistema operacional.

2.1. PostgreSQL

Site oficial: <https://www.postgresql.org/>

Linux: <https://www.youtube.com/watch?v=pqDNOGOcUks>

Windows: <https://www.youtube.com/watch?v=gHdnw-toYz8>

2.2. Node

Site oficial: <https://nodejs.org/en/>

Linux: <https://www.youtube.com/watch?v=AHWbz012kxI>

Windows: <https://www.youtube.com/watch?v=brSwmLQA0iA>

2.3. Gerenciador de pacote

A instalação do Node já vem com o gerenciador de pacote NPM, mas caso seja optado por utilizar o Yarn, seguir conforme links abaixo:

Linux: <https://www.hostinger.com.br/tutoriais/yarn-install/>

Windows: <https://yarnpkg.com/lang/pt-br/docs/install/#windows-stable>

2.4. API Client

O projeto não consta com interface gráfica, por esta razão se faz necessário um programa do tipo API Client para acessar suas funcionalidades. Neste documento detalharemos a configuração para o programa *Insomnia*, porém é possível utilizar qualquer outro programa de sua preferência:

Site oficial: <https://nodejs.org/en/>

Instalação (Linux e Windows): <https://support.insomnia.rest/article/23-installation>

3. Configurações

Os arquivos responsáveis pelas configurações do projeto, citados neste capítulo, ficam situados no seguinte endereço <Pasta raiz do projeto>/src/config.

3.1. Configurando a aplicação

O responsável pelas configurações da aplicação é o arquivo **app.js**, nele temos as seguintes opções:

- **Environment**; contêm o ambiente em que o projeto irá executar, aceitando os valor conforme abaixo:
 - development
 - test
 - production

É **importante** se atentar com o valor da variável, pois trata-se de um valor *case sensitive*.

- **AppPort**; define a porta em que a API será disponibilizada, no caso de não preenchimento será usado o padrão 3000.
- **Images Path**; define o local de armazenamento para as imagens que serão recebidas pela aplicação, no caso de não preenchimento será usado o padrão *<Pasta raiz do projeto>/uploads_*.

3.2. Configurando o banco de dados

O responsável pelas configurações da comunicação da API com o banco de dados do PostgreSQL está no arquivo **database.js**, nele teremos as seguintes opções editáveis:

- **host**; o endereço da base de dados (O padrão de instalação é localhost – 127.0.0.1)
- **username**; usuário utilizado no login para o banco de dados.
- **password**; senha utilizada no login para o banco de dados.
- **database**; base de dados no banco que será utilizada para armazenar as tabelas do projeto.

3.3. Configurando o Insomnia (API Client)

Após seguir os procedimentos de instalação do Insomnia, abra o aplicativo e siga o procedimento abaixo:

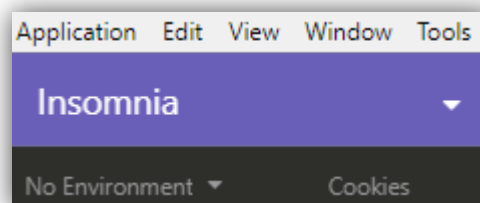


Figura 1- Clicar no **botão triângulo** para abertura do menu

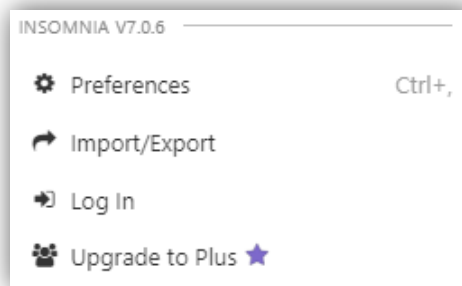


Figura 2- Clicar em **Import/Export**

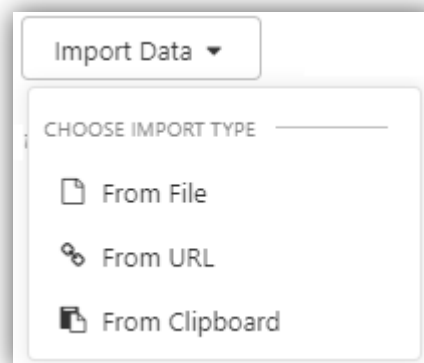


Figura 3- Acionar o botão **Import Data** e na sequência clicar em **FromFile**

Haverá a abertura de uma tela de seleção de arquivo com nome Import Insomnia Data, através dela, vá até <Pasta raiz do projeto>/docs, selecione o arquivo **Insomnia.json** e clique no botão **Import**. Será apresentada a mensagem abaixo:

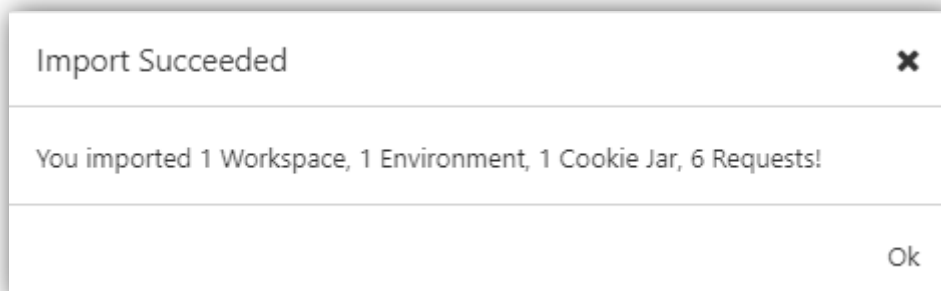


Figura 4- Após clicar em OK, a configuração já estará finalizada.

4. Scripts

4.1. Carregando as bibliotecas

Para puxar todas as bibliotecas utilizadas no projeto é necessário utilizar os comandos abaixo (Baseado no gerenciador de pacotes utilizado) na pasta raiz do projeto.

Yarn: yarn

Npm: npm install

Este processo carregará todas as bibliotecas e gerará a pasta **node_modules**.

4.2. Preparando a base de dados

4.2.1. Criando as tabelas

Para criar toda a estrutura da base de dados utilizada no projeto é necessário utilizar o seguinte comando na pasta raiz do projeto.

```
npx sequelize-cli db:migrate
```

Este processo gerará a estrutura de tabelas descritas no **capítulo 6** deste documento.

É **importante** se atentar que:

- já se tenha configurado a base de dados conforme o **capítulo 3.2**,
- garantir que a base de dados do campo **database** já esteja criado no banco de dados,
- que o servidor definido no campo **host** esteja disponível.

4.2.2. Criando os registros iniciais

Para criar todos os registros de referências em suas respectivas tabelas na base de dados utilizada no projeto é necessário utilizar o seguinte comando na pasta raiz do projeto

```
npx sequelize-cli db:seed:all
```

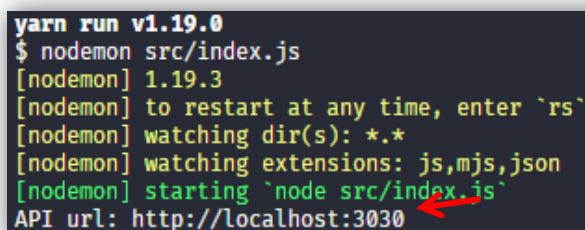
4.3. Rodando a aplicação

Para que a API comece a responder as chamadas é necessário utilizar os comandos abaixo (Baseado no gerenciador de pacotes utilizado) na pasta raiz do projeto.

Yarn: yarn start

Npm: npm run start

Quando iniciada, a API registra no console a url por onde está respondendo, conforme abaixo:



```
yarn run v1.19.0
$ nodemon src/index.js
[nodemon] 1.19.3
[nodemon] to restart at any time, enter `rs`
[nodemon] watching dir(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node src/index.js`
API url: http://localhost:3030
```

Figura 5- Apresentação da URL de resposta da API

4.4. Rodando os testes

Para executar os testes automatizados é necessário utilizar os comandos abaixo (Baseado no gerenciador de pacotes e sistema operacional utilizado) na pasta raiz do projeto. O relatório dos testes é apresentado no próprio console de comandos.

```
Test Suites: 7 passed, 7 total
Tests:      47 passed, 47 total
Snapshots:  0 total
Time:       6.971s
Ran all test suites.
Done in 8.56s.
```

Figura 6- Exemplo de relatório de testes

4.4.1. Linux

Yarn: yarn test-linux

Npm: npm run test-linux

4.4.2. Windows

Yarn: yarn test-windows

Npm: npm run test-windows

5. Organização das pastas do projeto

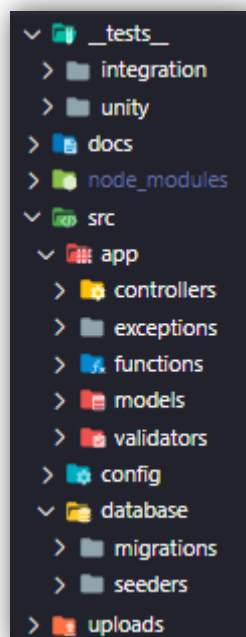


Figura 7- Estrutura de pastas do projeto

5.1. <Pasta raiz>

Pasta raiz do módulo, contendo todos os arquivos de configuração.

5.2. __tests__

Pasta com os arquivos de teste.

5.3. integration

Contém os testes que interagem com diversos módulos

5.4. unity

Contém os testes que interagem com um único módulo.

5.5. docs

Contém os documentos técnicos do projeto.

5.6. node_modules

Pasta com todas as bibliotecas do projeto.

5.7. src

Contendo todos os arquivos da API e, em sua raiz, os arquivos responsáveis pelo carregamento da API.

5.8. app

Contém os arquivos de lógica da aplicação.

5.9. controllers

Guarda os controladores da aplicação, responsáveis por orquestrar o fluxo entre as bibliotecas e funções.

5.10. exceptions

Contém os arquivos de exibição e tratamento de erro.

5.11. functions

Pasta com todos os arquivos de regras, validações e funcionalidades orquestradas pelos controladores.

5.12. models

Contém as definições e relacionamentos das entidades representadas por cada tabela do banco de dados.

5.13. validators

Contém os arquivos com a tipagem obrigatório das entradas pra API.

5.14. config

Contém as configurações internas da API.

5.15. database

Guarda os scripts de criação e preenchimento das tabelas no banco de dados.

5.16. migrations

Contém os scripts de criação das tabelas no banco de dados.

5.17. seeders

Contém os scripts de criação das tabelas no banco de dados.

5.18. uploads

Armazena as imagens de documentos que forem enviadas pela API.

6. Banco de dados da aplicação

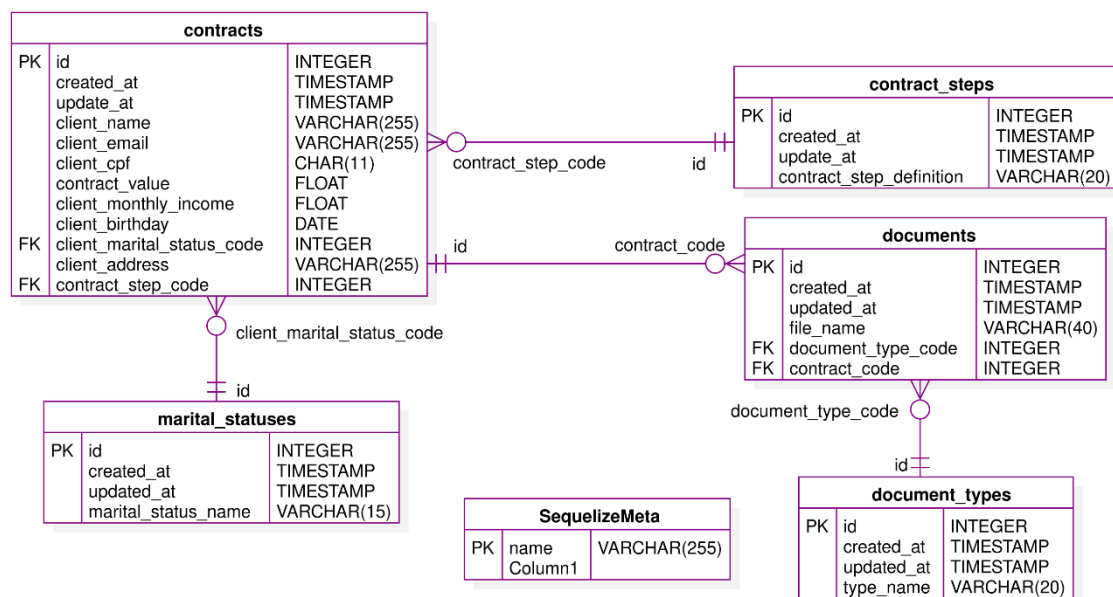


Figura 8- Modelo de entidade relacional

6.1. SequelizeMeta

Tabela responsável pelo armazenamento do histórico da criação de tabelas no banco de dados. Contém os seguintes campos:

name (Chave primária, varchar(max)) Nome do arquivo de criação da tabela que foi executado..

6.2. contracts

Tabela responsável pelo armazenamento dos contratos e etapa em cada um se encontra. Contém os seguintes campos:

id (Chave primária, integer) Código de identificação do contrato.
created_at (timestamp) Data da criação do contrato.
update_at (timestamp) Data da última alteração no contrato.
client_name (varchar(max)) Nome do cliente contratante.
client_email (varchar(max)) E-mail do cliente contratante.
client_cpf (char(11)) Número do CPF do cliente contratante.
contract_value (float) Valor do contrato.
client_monthly_income (float, default: 0) Renda mensal do cliente contratante.
client_birthday (date) Data de aniversário do cliente contratante.
client_marital_status_code (Chave estrangeira, integer) Código do estado civil do cliente contratante.
client_address (varchar(max)) Endereço residencial do cliente contratante.
contract_step_code (Chave estrangeira, integer) Código da etapa do andamento que o contrato se encontra.

6.3. documents

Tabela responsável pelo armazenamento da correspondência entre as imagens recebidas, documentos do cliente contratante e o contrato respectivo. Contém os seguintes campos:

id (Chave primária, integer) Código de identificação da etapa.
created_at (timestamp) Data da criação do registro.
update_at (timestamp) Data da última alteração no registro.
file_name (char(40)) Nome do arquivo armazenado referente ao documento.
document_type_code (Chave estrangeira, integer) Tipo do documento da imagem.

6.1. document_types

Tabela responsável pelo de-para dos tipos de documentos com um descritivo de cada um deles. Contém os seguintes campos:

id (Chave primária, integer) Código de identificação do tipo de documento.
created_at (timestamp) Data da criação do registro.
update_at (timestamp) Data da última alteração no registro.
type_name (varchar(20)) Nome do documento referente.

6.2. contract_steps

Tabela responsável pelo de-para dos códigos de etapas do contrato com um descritivo de cada uma delas. Contém os seguintes campos:

id (Chave primária, integer) Código de identificação da etapa.
created_at (timestamp) Data da criação do registro.
update_at (timestamp) Data da última alteração no registro.
contract_step_definition (varchar(20)) Definição da etapa.

6.3. marital_statuses

Tabela responsável pelo de-para dos estados civis do cliente contratante com um descritivo de cada um deles. Contém os seguintes campos:

id (Chave primária, integer) Código de identificação do estado civil.
created_at (timestamp) Data da criação do registro.

update_at (timestamp) Data da última alteração no registro.
marital_status_name (varchar(15)) Definição da etapa.

7. Endpoints

7.1. Retorna a lista com os tipos de documento .

GET /listDocumentTypes

Retorno:

```
[{
  "id": 1,
  "type_name": "Tipo do documento",
  "created_at": "9999-99-99T99:99:13.494Z",
  "updated_at": "9999-99-99T99:1:13.494Z"
}]
```

7.2. Retorna a lista de estados civis .

GET /listMaritalStatus

Retorno:

```
[{
  "id": 1,
  "marital_status_name": "Nome do estado civil",
  "created_at": "9999-99-99T99:99:13.494Z",
  "updated_at": "9999-99-99T99:1:13.494Z"
}]
```

7.3. Retorna a lista de etapas de um contrato.

GET /contract/listSteps

Retorno:

```
[{
  "id": 1,
  "contract_step_definition": "Nome do passo",
  "created_at": "9999-99-99T99:99:13.494Z",
  "updated_at": "9999-99-99T99:1:13.494Z"
}]
```

7.4. Retorna o contrato correspondente a seu código de identificação.

GET /contract/listById/:id

Parâmetros:

- **:id** ; Código de identificação do contrato.

Retorno:

```
{
  "id": 9,
  "client_name": "Nome do cliente",
  "client_email": "Email do cliente",
  "client_cpf": "999999999999",
  "contract_value": 9.9,
  "client_monthly_income": 9,
  "client_birthday": 9999/99/99,
  "client_marital_status_code": 9,
  "client_address": "Endereço do cliente",
  "contract_step_code": 9,
  "created_at": "9999-99-99T99:99:13.494Z",
  "updated_at": "9999-99-99T99:1:13.494Z"
}
```

7.5. Lista os contratos vinculados a um cliente pelo seu CPF

GET /contract/listByCpf/:cpf

Parâmetros:

- **:cpf** ; Cpf do cliente a ser pesquisado.

Retorno:

```
[{
  "id": 9,
  "client_name": "Nome do cliente",
  "client_email": "Email do cliente",
  "client_cpf": "99999999999",
  "contract_value": 9.9,
  "client_monthly_income": 9,
  "client_birthday": 9999/99/99,
  "client_marital_status_code": 9,
  "client_address": "Endereço do cliente",
  "contract_step_code": 9,
  "created_at": "9999-99-99T99:99:13.494Z",
  "updated_at": "9999-99-99T99:1:13.494Z"
}]
```

7.6. Cria um novo contrato de empréstimo

POST /contract/new

Padrão do body esperado: Json

Exemplo de body esperado:

```
{
  "client_name": "Nome do cliente",
  "client_email": "Email do cliente",
  "client_cpf": "99999999999",
  "contract_value": 9.9,
  "client_monthly_income": 9,
  "client_birthday": 9999-99-99,
  "client_marital_status_code": 9,
  "client_address": "Endereço do cliente",
}
```

Retorno:

```
{
  "id": 9,
  "client_name": "Nome do cliente",
  "client_email": "Email do cliente",
  "client_cpf": "99999999999",
  "contract_value": 9.9,
  "client_monthly_income": 9,
  "client_birthday": 9999-99-99,
  "client_marital_status_code": 9,
  "client_address": "Endereço do cliente",
  "contract_step_code": 9,
  "created_at": "9999-99-99T99:99:13.494Z",
  "updated_at": "9999-99-99T99:1:13.494Z"
}
```

7.7. Altera um contrato de empréstimo já existente

PUT /contract/update/:id

Parâmetros:

- **:id** ; Código de identificação do contrato.

Padrão do body esperado: Json

Exemplo de body esperado:

```
{
  "client_name": "Nome do cliente",
  "client_email": "Email do cliente",
  "client_cpf": "99999999999",
  "contract_value": 9.9,
  "client_monthly_income": 9,
  "client_birthday": 9999/99/99,
  "client_marital_status_code": 9,
  "client_address": "Endereço do cliente",
}
```


Retorno:

```
{
  "id": 9,
  "client_name": "Nome do cliente",
  "client_email": "Email do cliente",
  "client_cpf": "999999999999",
  "contract_value": 9.9,
  "client_monthly_income": 9,
  "client_birthday": 9999/99/99,
  "client_marital_status_code": 9,
  "client_address": "Endereço do cliente",
  "contract_step_code": 9,
  "created_at": "9999-99-99T99:99:13.494Z",
  "updated_at": "9999-99-99T99:1:13.494Z"
}
```

7.8. Envia documentos referentes a um contrato de empréstimo

POST /contract/sendDocument/:id?type=:idType

Parâmetros:

- **:id** ; Código de identificação do contrato.
- **:idType** ; Código de identificação do tipo do documento a ser enviado.

Padrão de body esperado: Multipart Form

Exemplo de body esperado:

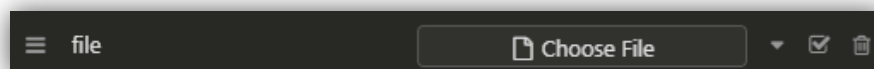


Figura 9- Body para envio de arquivo por multipart form

Retorno:

```
{
  "id": 1,
  "file_name": "hash.jpg",
  "document_type_code": 1,
  "contract_code": 1,
  "created_at": "9999-99-99T99:99:13.494Z",
  "updated_at": "9999-99-99T99:1:13.494Z"
}
```

7.9. Aprova ou desaprova o contrato de empréstimo

POST /contract/approval/:id

Parâmetros:

- :id ; Código de identificação do contrato.

Exemplo de body esperado:

```
{
  "approval": true
}
```

```
{
  "approval": false
}
```

Retorno:

```
{
  "id": 9,
  "client_name": "Nome do cliente",
  "client_email": "Email do cliente",
  "client_cpf": "99999999999",
  "contract_value": 9.9,
  "client_monthly_income": 9,
  "client_birthday": 9999/99/99,
  "client_marital_status_code": 9,
  "client_address": "Endereço do cliente",
  "contract_step_code": 9,
  "created_at": "9999-99-99T99:99:13.494Z",
  "updated_at": "9999-99-99T99:1:13.494Z"
}
```

8. Melhorias

Infelizmente, não consegui concluir tudo que esperava implantar neste projeto até a data deste documento. Listo abaixo as features e melhorias que ficaram pendentes:

- Criar uma estrutura de container docker (<https://www.docker.com/>) pra aplicação, visando facilitar a montagem de ambiente durante avaliação.
- Adicionar métricas com o Prometheus (<https://prometheus.io/>) para acompanhar tempo de processamento, erros, horários de pico, etc.
- Utilizar Grafana (<https://grafana.com/>) para criação de gráficos dos dados gerados pelo Prometheus.
- Utilizaria o Sentry (<https://sentry.io/welcome/>) para o melhor acompanhamento dos erros pontuais não abordados pelo Prometheus.

- Utilizar o Swagger (<https://swagger.io/>) para uma documentação mais dinâmica dos endpoints.
- Melhorar a cobertura de testes
- Adicionar as imagens dos documentos na S3 da Amazon (<https://aws.amazon.com/pt/s3/>) ou similares.
- Migrar para arquitetura serverless
- Utilizar o método de **eager load** no carregamento das listagens, já preenchendo com os dados correspondentes as chaves estrangeiras da tabela carregada.
- Permitir exclusão de imagens de documentos relacionadas aos contratos.
- Adicionar erros no status HTTP das respostas dos Endpoints para facilitar a identificação.